EXP NO: 1	Install Apache Hadoop
Date:	

AIM: To Install Apache Hadoop software on Windows.

Hadoop software can be installed in three modes of Hadoop is a Java-based programming framework that supports the processing and storage of extremely large datasets on a cluster of inexpensive machines. It was the first major open-source project in the big data playing field and is sponsored by the Apache Software Foundation.

Hadoop-2.7.3 is comprised of four main layers:

- **Hadoop Common** is the collection of utilities and libraries that support other Hadoop modules.
- **HDFS**, which stands for Hadoop Distributed File System, is responsible for persisting data to disk.
- **YARN**, short for Yet Another Resource Negotiator, is the "operating system" for HDFS.
- **MapReduce** is the original processing model for Hadoop clusters. It distributes work within the cluster or map, then organizes and reduces the results from the nodes into a response to a query. Many other processing models are available for the 2.x version of Hadoop.

Hadoop clusters are relatively complex to set up, so the project includes a stand-alone mode which is suitable for learning about Hadoop, performing simple operations, and debugging.

Procedure:

we'll install Hadoop in stand-alone mode and run one of the example MapReduce programs it includes to verify the installation.

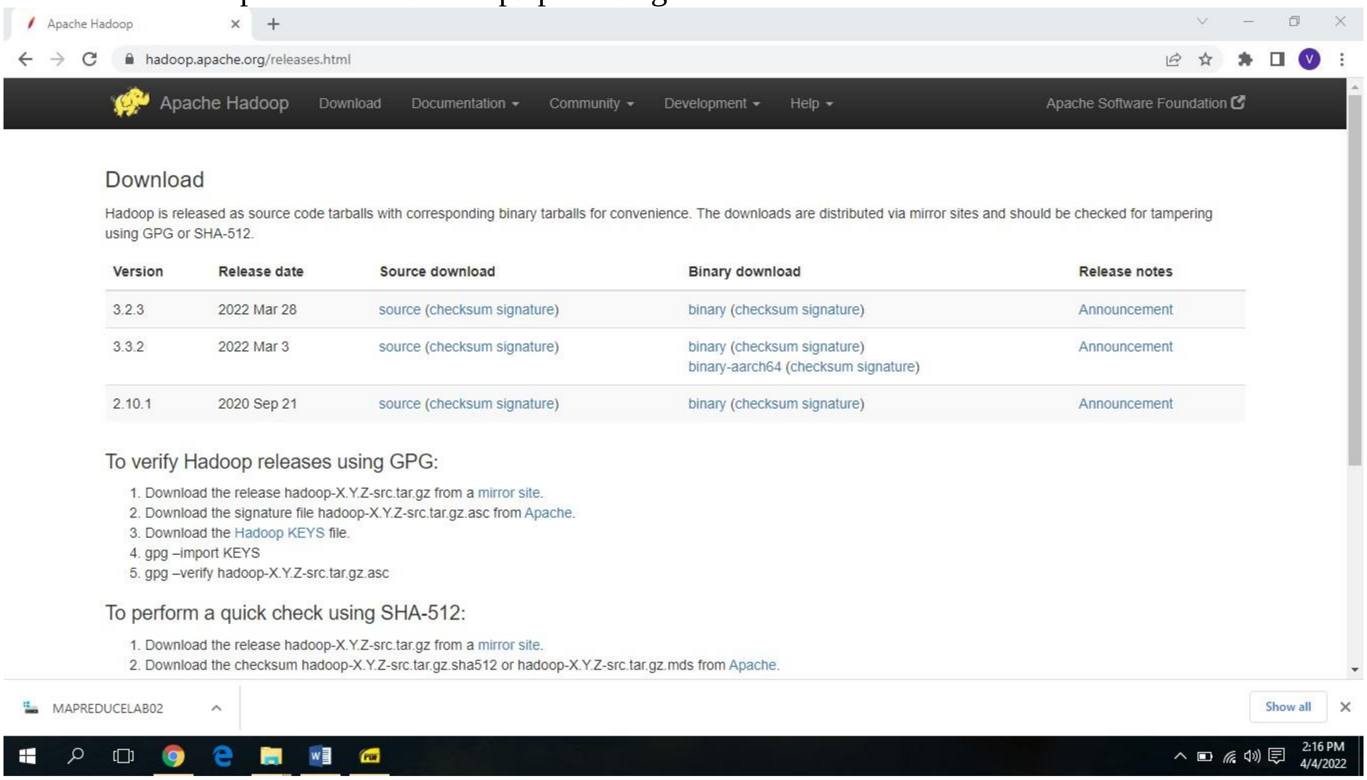
Prerequisites:

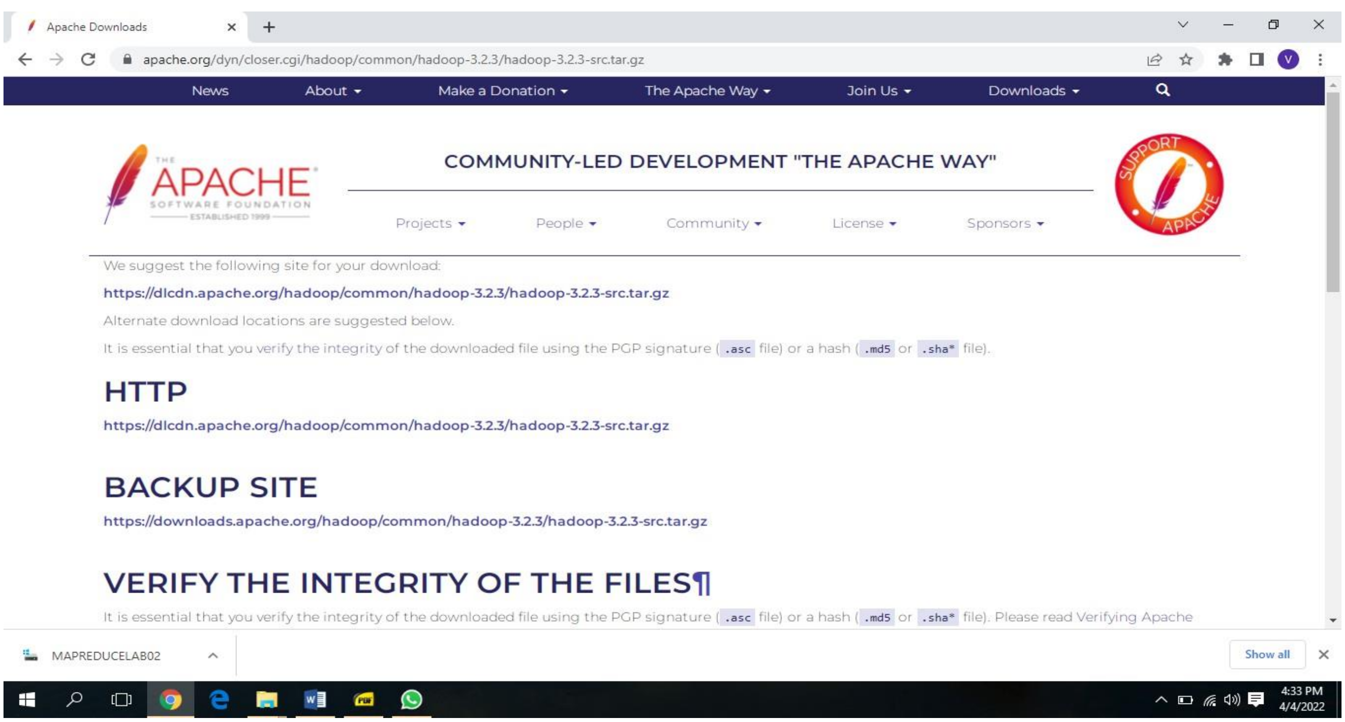
Step1: Installing Java 8 version.

Openjdk version "1.8.0_91"
OpenJDK Runtime Environment (build 1.8.0_91-8u91-b14-3ubuntu1~16.04.1-b14)
OpenJDK 64-Bit Server VM (build 25.91-b14, mixed mode)
This output verifies that OpenJDK has been successfully installed.
Note: To set the path for environment variables. i.e. JAVA_HOME

Step2: Installing Hadoop
With Java in place, we'll visit the Apache Hadoop Releases page to find the most recent stable release. Follow the binary for the current release:

Download Hadoop from www.hadoop.apache.org





Procedure to Run Hadoop

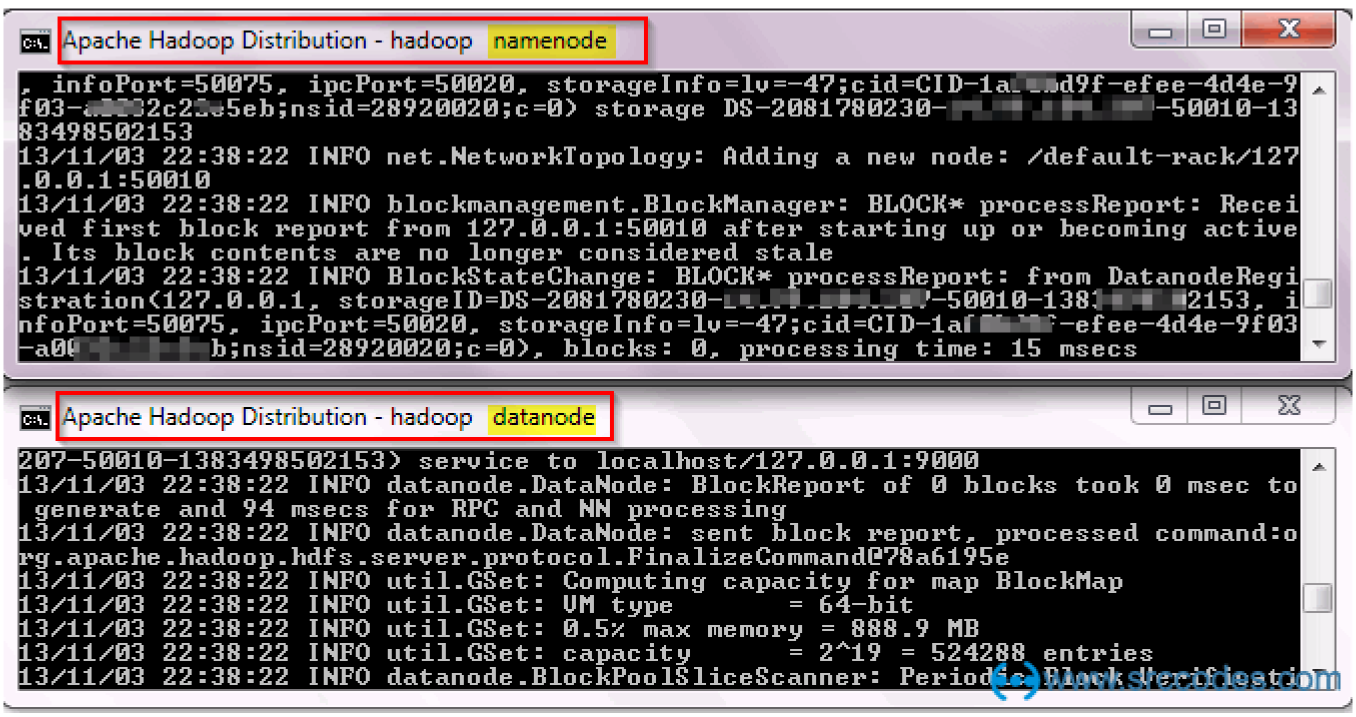
1. Install Apache Hadoop 2.2.0 in Microsoft Windows OS

If Apache Hadoop 2.2.0 is not already installed then follow the post Build, Install, Configure and Run Apache Hadoop 2.2.0 in Microsoft Windows OS.

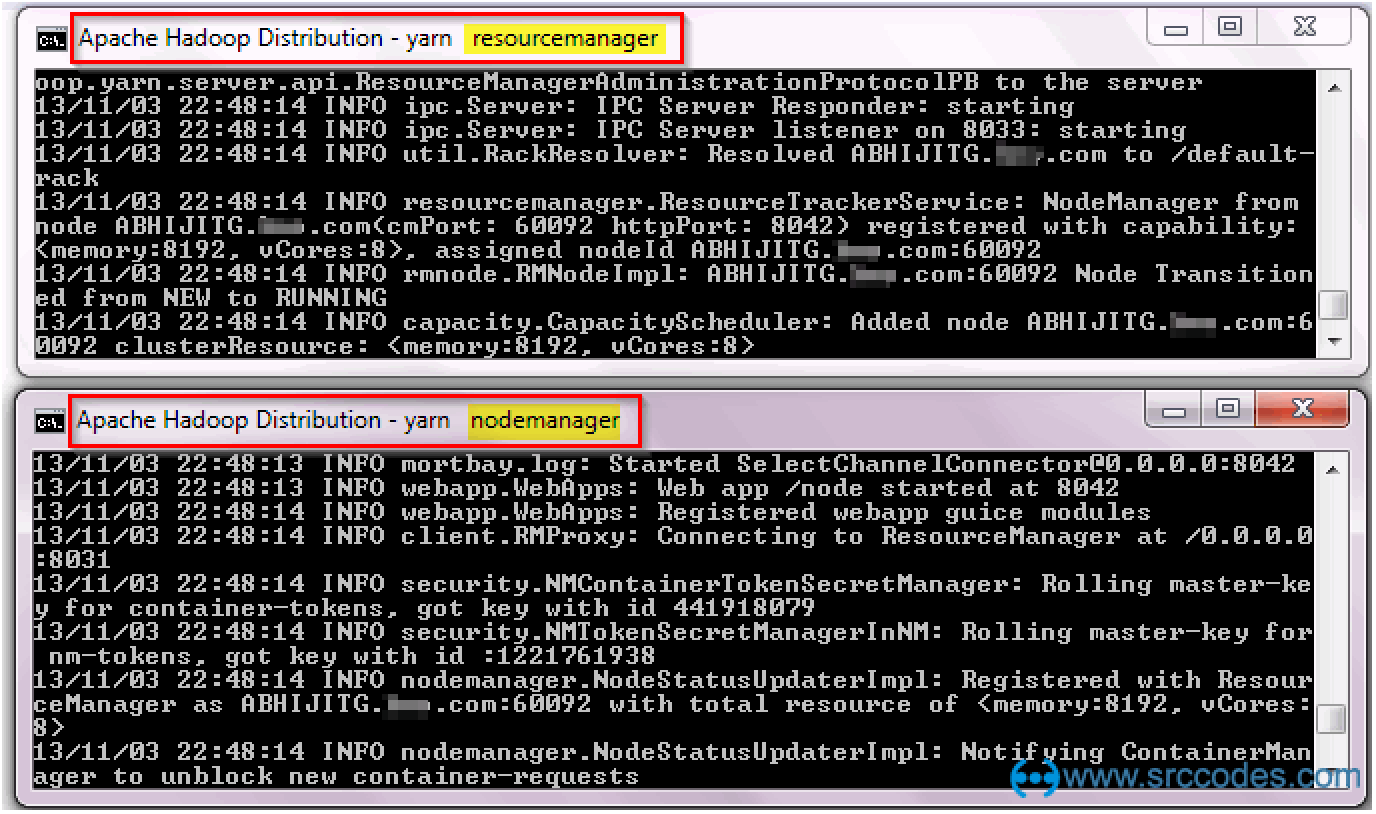
2. Start HDFS (Namenode and Datanode) and YARN (Resource Manager and Node Manager)

Run following commands.
Command Prompt
C:\Users\abhijitg>cd c:\hadoop
c:\hadoop>sbin\start-dfs
c:\hadoop>sbin\start-yarn
starting yarn daemons

Namenode, **Datanode**, **Resource Manager** and **Node Manager** will be started in few minutes and ready to execute Hadoop **MapReduce** job in the Single Node (pseudo-distributed mode) cluster.



Resource Manager & Node Manager:



Run wordcount MapReduce job

Now we'll run **wordcount** MapReduce job available in

%HADOOP_HOME%\share\hadoop\mapreduce\hadoop-mapreduce-examples- 2.2.0.jar

Create a text file with some content. We'll pass this file as input to the **wordcount** MapReduce job for counting words.
C:\file1.txt

```
Install Hadoop

Run Hadoop Wordcount Mapreduce Example
```

Create a directory (say 'input') in HDFS to keep all the text files (say 'file1.txt') to be used for counting words.

C:\Users\abhijitg>cd c:\hadoop
C:\hadoop>bin\hdfs dfs -mkdir input

Copy the text file (say 'file1.txt') from local disk to the newly created 'input' directory in HDFS

C:\hadoop>bin\hdfs dfs -copyFromLocal c:/file1.txt input

Check content of the copied file.

C:\hadoop>hdfs dfs -ls input

Found 1 items

-rw-r--r-- 1 ABHIJITG supergroup 55 2014-02-03 13:19 input/file1.txt

C:\hadoop>bin\hdfs dfs -cat input/file1.txt

Install Hadoop
Run Hadoop Wordcount Mapreduce Example
Run the wordcount MapReduce job provided
in %HADOOP_HOME%\share\hadoop\mapreduce\hadoop-mapreduce-examples-2.2.0.jar
C:\hadoop>bin\yarn jar share/hadoop/mapreduce/hadoop-mapreduce-examples- 2.2.0.jar wordcount input output

14/02/03 13:22:02 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
14/02/03 13:22:03 INFO input.FileInputFormat: Total input paths to process: 1 14/02/03 13:22:03 INFO mapreduce.JobSubmitter: number of splits:1
:
:
14/02/03 13:22:04 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1391412385921_0002
14/02/03 13:22:04 INFO impl.YarnClientImpl: Submitted application application_1391412385921_0002 to ResourceManager at /0.0.0.0:8032 14/02/03 13:22:04 INFO mapreduce.Job: The url to track the job: http://ABHIJITG:8088/proxy/application_1391412385921_0002/
14/02/03 13:22:04 INFO mapreduce.Job: Running job: job_1391412385921_0002 14/02/03 13:22:14 INFO mapreduce.Job: Job job_1391412385921_0002 running in uber mode: false
14/02/03 13:22:14 INFO mapreduce.Job: map 0% reduce 0%
14/02/03 13:22:22 INFO mapreduce.Job: map 100% reduce 0%
14/02/03 13:22:30 INFO mapreduce.Job: map 100% reduce 100%
14/02/03 13:22:30 INFO mapreduce.Job: Job job_1391412385921_0002 completed successfully
14/02/03 13:22:31 INFO mapreduce.Job: Counters: 43 File
System Counters
FILE: Number of bytes read=89
FILE: Number of bytes written=160142
FILE: Number of read operations=0 FILE: Number of large read operations=0 FILE: Number of write operations=0
HDFS: Number of bytes read=171
HDFS: Number of bytes written=59
HDFS: Number of read operations=6
HDFS: Number of large read operations=0 HDFS: Number of write operations=2
Job Counters
Launched map tasks=1

Launched reduce tasks=1
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=5657 Total time spent by all reduces in occupied slots (ms)=6128

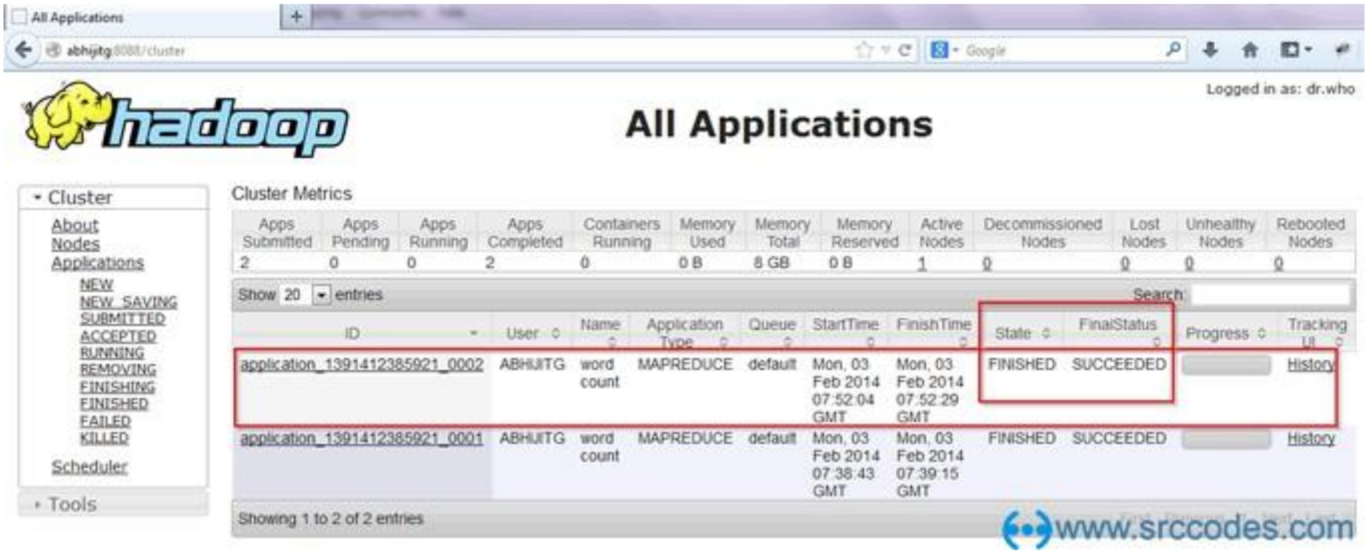
Map-Reduce Framework Map
input records=2 Map
output records=7 Map
output bytes=82
Map output materialized bytes=89 Input
split bytes=116
Combine input records=7
Combine output records=6
Reduce input groups=6
Reduce shuffle bytes=89
Reduce input records=6
Reduce output records=6
Spilled Records=12 Shuffled
Maps =1
Failed Shuffles=0 Merged
Map outputs=1
GC time elapsed (ms)=145 CPU
time spent (ms)=1418
Physical memory (bytes) snapshot=368246784 Virtual
memory (bytes) snapshot=513716224 Total committed
heap usage (bytes)=307757056

Shuffle Errors
BAD_ID=0 CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0

File Input Format Counters Bytes
Read=55

File Output Format Counters
Bytes Written=59

http://abhijitg:8088/cluster



Result: We has been successfully installed Hadoop in stand-alone mode and verified it by running an example program which is provided.

EXP NO: 2	MapReduce program to calculate the frequency
Date:	

AIM: To Develop a MapReduce program to calculate the frequency of a given word in a given file **Map Function** – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

Example – (Map function in Word Count)

Input

Set of data

Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN, BUS, buS, caR, CAR, car, BUS, TRAIN

Output

Convert into another set of data

(Key,Value)

(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1), (BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Reduce Function – Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

Example – (Reduce function in Word Count)

Input Set of Tuples

(output of Map function)

(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1), (BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Output Converts into smaller set of tuples

(BUS,7), (CAR,7), (TRAIN,4)

Workflow of Program

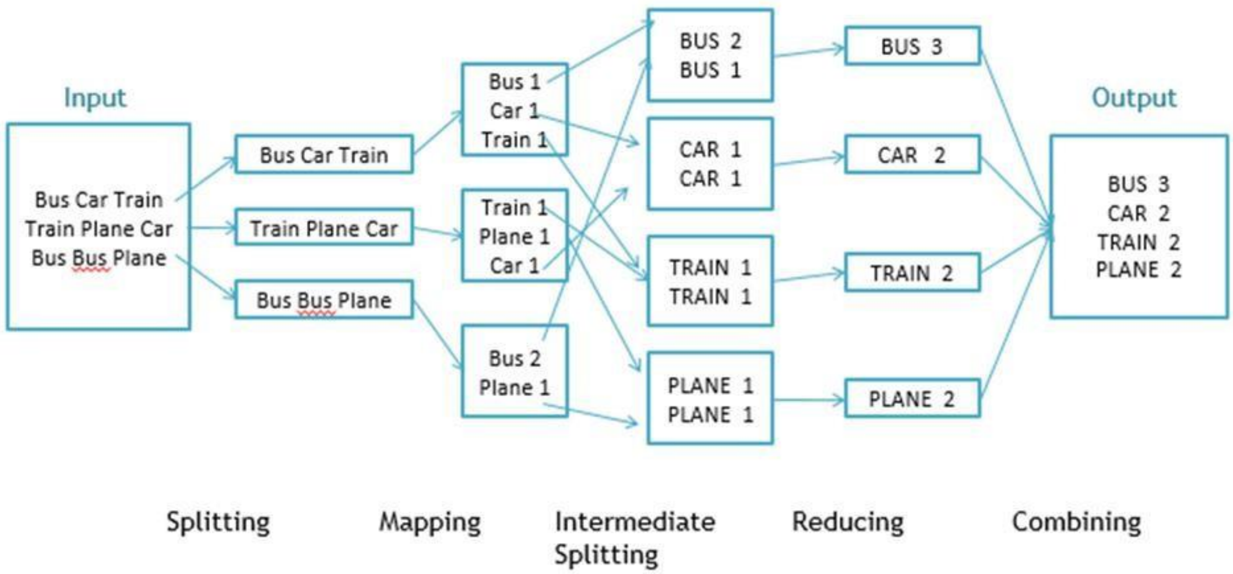


Fig. WorkFlow of MapReducing

Workflow of MapReduce consists of 5 steps

1. **Splitting** – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line ('\n').
2. **Mapping** – as explained above
3. Intermediate splitting – the entire process in parallel on different clusters. In order to group them in "Reduce Phase" the similar KEY data should be on same cluster.
4. **Reduce** – it is nothing but mostly group by phase
5. **Combining** – The last phase where all the data (individual result set from each cluster) is combined together to form a Result

Now Let's See the Word Count Program in Java

Make sure that Hadoop is installed on your system with java idk Steps to follow

Step 1. Open Eclipse> File > New > Java Project > (Name it – MRProgramsDemo) > Finish
Step 2. Right Click > New > Package (Name it - PackageDemo) > Finish
Step 3. Right Click on Package > New > Class (Name it - WordCount)
Step 4. Add Following Reference Libraries –

Right Click on Project > Build Path> Add External Archivals

- /usr/lib/hadoop-0.20/hadoop-core.jar
- Usr/lib/hadoop-0.20/lib/Commons-cli-1.2.jar

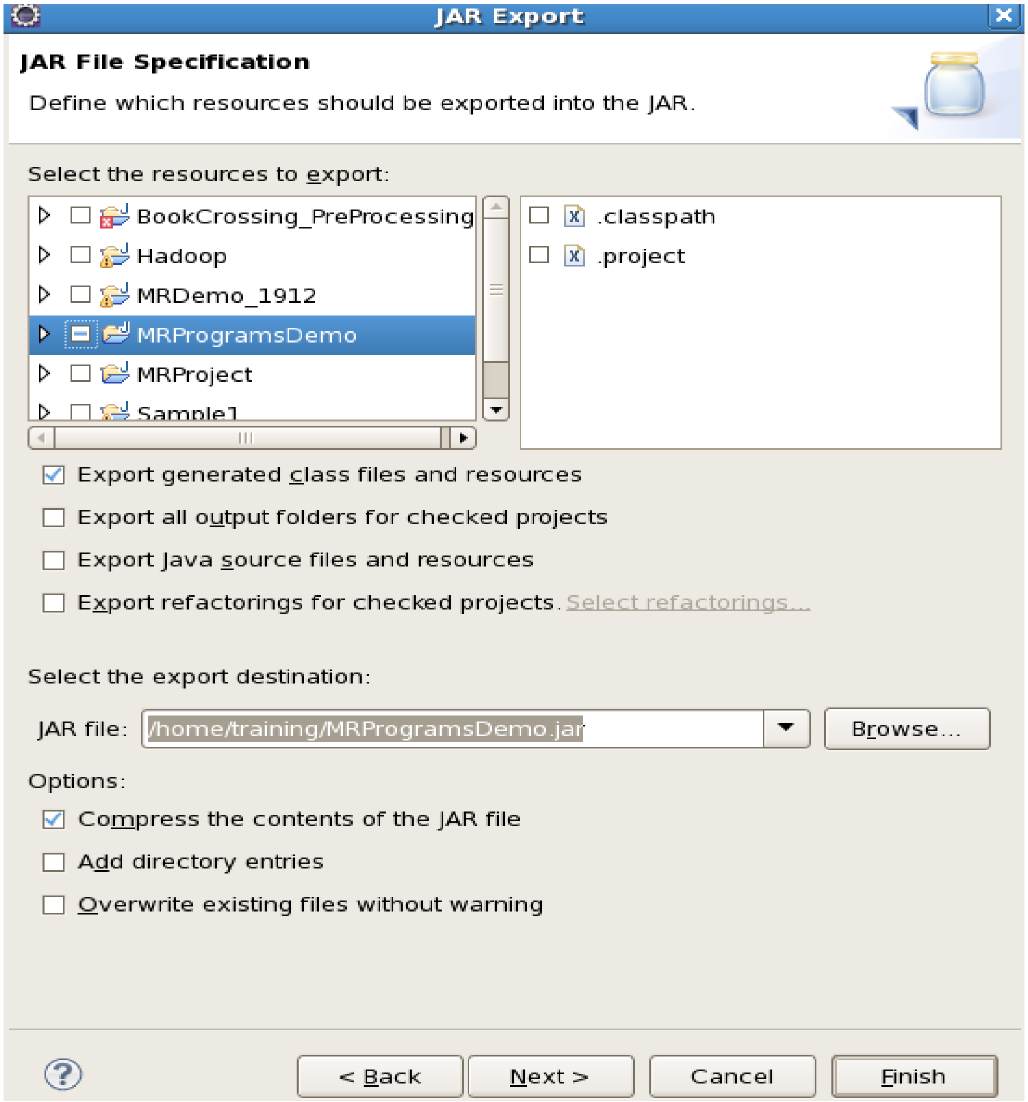
Program: Step 5. Type following Program:

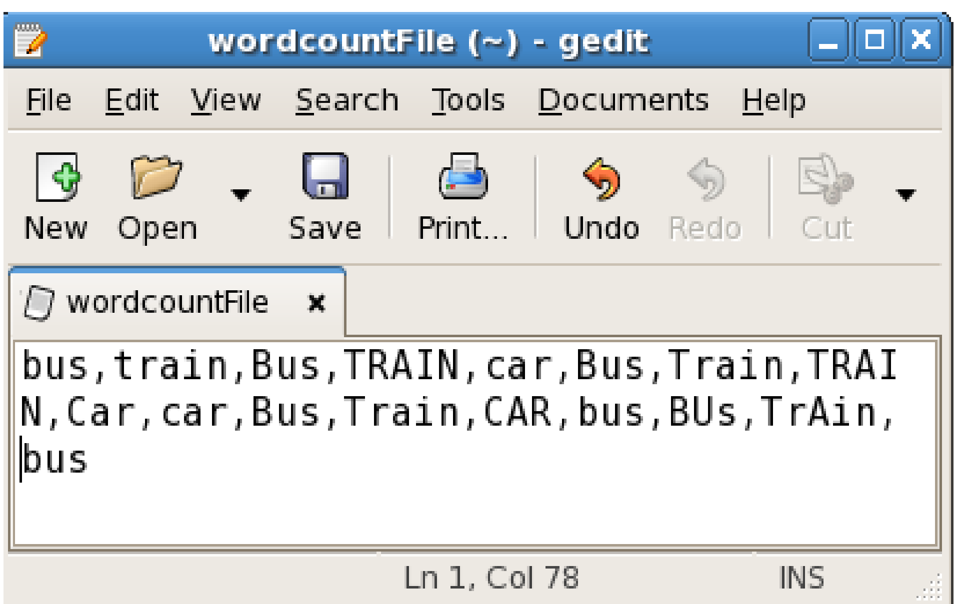
```
package PackagedEmo; import
java.io.IOException;
import          org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import         org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {
public static void main (String [] args) throws Exception
{
Configuration c=new Configuration ();
String [] files=new GenericOptionsParser(c,args).getRemainingArgs();

Path input=new Path (files [0]);
Path output=new Path (files [1]);
Job     j=new     Job(c,"wordcount");
j.setJarByClass(WordCount.class);
j.setMapperClass(MapForWordCount.class);
j.setReducerClass(ReduceForWordCount.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(j,                          input);
FileOutputFormat.setOutputPath(j,                        output);
System.exit(j.waitForCompletion(true)?0:1);
}
public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable> {

public void map (LongWritable key, Text value, Context con) throws IOException,
InterruptedException


{
String line = value.toString();
String [] words=line.split(",");
```

```
for (String word: words)
{
   Text  outputKey  =  new  Text(word.toUpperCase().  trim  ());
   IntWritable      outputValue      =      new      IntWritable(1);
   con.write(outputKey, outputValue);
}
}
}

public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
{
public void reduces (Text word, Iterable<IntWritable> values, Context con) throws IOException,

InterruptedException
{
int sum = 0;
for (IntWritable value: values)
{
sum += value.get();
}
con.write(word, new IntWritable(sum));
}
}
}
```

Make Jar File

Right Click on Project> Export> Select export destination as Jar File > next> Finish


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
BookCrossing_PreProcessing
Hadoop
MRDemo_1912
MRProgramsDemo
MRProject
Sample1
.classpath
.project
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects. Select refactorings...
Select the export destination:
JAR file: /home/training/MRProgramsDemo.jar
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Cancel
Finish


wordcountFile (~) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut
wordcountFile
bus,train,Bus,TRAIN,car,Bus,Train,TRAI
N,Car,car,Bus,Train,CAR,bus,BUs,TrAin,
bus
Ln 1, Col 78
INS

To Move this into Hadoop directly, open the terminal and enter the following commands:

[training@localhost ~] $ hadoop fs -put wordcountFile wordCountFile

Run Jar file

(Hadoop jar jarfilename.jar packageName.ClassName PathToInputTextFile PathToOutputDirectry)

[training@localhost ~] $ Hadoop jar MRProgramsDemo.jar PackageDemo.WordCount wordCountFile MRDir1

Result: Open Result

[training@localhost ~] $ hadoop fs -ls MRDir1

Found 3 items
-rw-r--r-- 1 training supergroup
0 2016-02-23 03:36 /user/training/MRDir1/_SUCCESS
drwxr-xr-x - training supergroup
0 2016-02-23 03:36 /user/training/MRDir1/_logs
-rw-r--r-- 1 training supergroup
20 2016-02-23 03:36 /user/training/MRDir1/part-r-00000

[training@localhost ~] $ hadoop fs -cat MRDir1/part-r-00000

BUS 7
CAR 4
TRAIN 6

Result: MapReduce program to calculate the frequency is executed successfully.

EXP NO: 3	Implement MapReduce program that processes a weather dataset
Date:	

AIM: The aim is to Implement MapReduce program that processes a weather dataset.

Procedure:

- The code simulates weather data with random temperature and humidity values.
- It defines map functions to categorize temperature and humidity data into key-value pairs.
- A reduce function aggregates the mapped data by summing up the values for each key.
- The MapReduce function combines mapping and reducing operations:
    - It maps the data using a specified mapper function.
    - It groups the mapped data by keys.
    - It reduces each group using a reducer function.
- In the main execution:
    - Simulated weather data is generated.
    - MapReduce is performed separately for temperature and humidity.
    - The counts of temperature and humidity values are printed as output.

Program:

```
import random
from multiprocessing import Pool

# Simulated weather data generator
def generate_weather_data(num_records):
    weather_data = []
    for _ in range(num_records):
        temperature = random.randint(-20, 40)
        humidity = random.randint(0, 100)
        weather_data.append((temperature, humidity))
    return weather_data
# Map function to process temperature data
def map_temperature(data):
    temperature, humidity = data
    return temperature, 1
#Map function to process humidity data
def map_humidity(data):
    temperature, humidity = data
    return humidity, 1
# Reduce function to aggregate counts
def reduce_counts(data):
    key, counts = data
```

```
    return key, sum(counts)
# MapReduce function


def map_reduce(data, mapper, reducer):
    mapped_data = [mapper(item) for item in data]
    grouped_data = {}
    for key, value in mapped_data:
        grouped_data.setdefault(key, []). append(value)
    reduced_data = [reducer ((key, value)) for key, value in grouped_data.items()]
    return reduced_data

if __name__ == '__main__':
    # Simulate weather dataset
    weather_data = generate_weather_data(1000)

    # Run MapReduce for temperature
    temperature_counts = map_reduce(weather_data, map_temperature, reduce_counts)
    print ("Temperature counts:")
    print(temperature_counts)

    # Run MapReduce for humidity
    humidity_counts = map_reduce(weather_data, map_humidity, reduce_counts)
    print ("Humidity counts:")
    print(humidity_counts)
```

OUTPUT:

Temperature counts:

[(-8, 15), (22, 18), (30, 13), (4, 18), (15, 12), (36, 17), (17, 17), (-13, 20), (39, 18), (3, 13), (27, 13), (-2, 12), (7, 18), (0, 15), (-16, 15), (-20, 20), (-9, 22), (16, 22), (28, 16), (40, 15), (23, 13), (-11, 19), (1, 24), (2, 24), (8, 23), (-18, 24), (-19, 16), (11, 17), (-10, 26), (-7, 17), (19, 15), (-4, 12), (6, 21), (-3, 16), (31, 15), (-14, 14), (12, 20), (-6, 19), (18, 10), (26, 13), (5, 9), (-1, 15), (29, 14), (20, 19), (-12, 14), (32, 13), (-15, 18), (9, 22), (14, 15), (38, 13), (13, 21), (33, 20), (25, 13), (35, 16), (10, 11), (37, 18), (21, 14), (24, 16), (34, 15), (-17, 7), (-5, 10)]

Humidity counts:

[(27, 10), (49, 9), (98, 13), (5, 10), (86, 12), (43, 7), (42, 10), (54, 11), (62, 8), (77, 16), (12, 13), (55, 16), (65, 16), (70, 17), (45, 8), (83, 6), (0, 10), (52, 7), (66, 8), (4, 11), (74, 13), (61, 10), (13, 16), (48, 13), (6, 4), (87, 8), (99, 8), (8, 8), (79, 8), (80, 6), (91, 10), (16, 10), (30, 15), (89, 11), (20, 12), (46, 13), (56, 7), (69, 7), (60, 7), (40, 14), (63, 12), (14, 10), (58, 10), (57, 13), (71, 7), (85, 7), (35, 6), (51, 12), (9, 9), (97, 7), (17, 13), (18, 13), (32, 8), (28, 15), (50, 8), (47, 9), (78, 11), (29, 5), (100, 9), (96, 8), (92, 13), (37, 9), (53, 11), (76, 13), (75, 10), (31, 14), (2, 16), (68, 14), (34, 7), (94, 10), (10, 8), (39, 10), (90, 9), (64, 7), (1, 9), (7, 10), (33, 15), (21, 5), (26, 6), (81, 8), (15, 7), (72, 13), (23, 15), (93, 5), (82, 13), (95, 10), (59, 9), (88, 8), (24, 11), (19, 13), (36, 6), (41, 8), (11, 8), (22, 6), (44, 10), (84, 3), (73, 9), (3, 7), (25, 9), (38, 9), (67, 7)]

Result: Implementing MapReduce program that processes a weather dataset is executed successfully.

EXP NO: 4	Collect sensor data from any real time application and apply preprocessing techniques
Date:	

Aim: The aim is to Collect sensor data from any real time application and apply preprocessing techniques.

Procedure:
Preprocessing sensor data is a crucial step in preparing it for further analysis or machine learning. Let’s walk through the process using Python:

1. **Import Necessary Libraries**: First, import the required libraries such as Pandas, NumPy, and Scikit-Learn. These will help you manipulate and preprocess the data effectively

2. **Python**

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

3. **Load the Dataset**: Load your sensor data into a Pandas DataFrame. For example, if you have a CSV file, you can read it like this:

   Python

```
df = pd.read_csv('path/to/your/sensor_data.csv')
print(df.head())
```

   This will display the first few rows of your dataset.

4. **Data Cleaning and Preprocessing**:
   - Handle missing values: Identify and handle any missing data (e.g., replace with mean, median, or drop rows/columns).
   - Remove irrelevant columns: Drop any columns that aren’t useful for your analysis.
   - Convert data types: Ensure that data types are appropriate for each feature (e.g., numeric, categorical).

5. **Feature Scaling**: Normalize or standardize your features to bring them to a similar scale. For example, use Min-Max scaling:

6. **Exploratory Data Analysis (EDA)**: Visualize your data using libraries like Seaborn and Matplotlib. Explore relationships between features and identify outliers.

7. **Feature Engineering**: Create new features if needed. For instance, derive additional features from existing ones (e.g., ratios, averages).

8. **Handling Categorical Variables**: If your data contains categorical variables, encode them.

9. **Split Data into Training and Test Sets**: Divide your dataset into training and test subsets for model evaluation.

Code:

```
import random

# Function to generate a simple weather dataset
def generate_weather_data(num_records):
    weather_data = []
    for _ in range(num_records):
        temperature = random.randint(-20, 40) # Temperature in Celsius
        humidity = random.randint(0, 100)      # Humidity in percentage
        weather_data.append((temperature, humidity))
    return weather_data

# Function to apply preprocessing techniques
def preprocess(data):
    preprocessed_data = []
    for temperature, humidity in data:
        # Example preprocessing: Filtering out temperatures below 0
        if temperature >= 0:
            # Example preprocessing: Normalizing humidity to range [0, 1]
            humidity_normalized = humidity / 100.0
            preprocessed_data.append((temperature, humidity_normalized))
    return preprocessed_data

if __name__ == '__main__':
    # Generate a simple weather dataset
    weather_data = generate_weather_data(1000)

    # Apply preprocessing techniques
    preprocessed_data = preprocess(weather_data)

    # Print preprocessed data
    print ("Preprocessed Weather Data:")
    for temperature, humidity in preprocessed_data:
        print (f"Temperature: {temperature}°C, Humidity: {humidity}")

    # Additional preprocessing or analysis can be performed here
```

OUTPUT:

Preprocessed Weather Data:
Temperature: 37°C, Humidity: 0.68
Temperature: 39°C, Humidity: 0.31
Temperature: 33°C, Humidity: 0.76
Temperature: 24°C, Humidity: 0.88
Temperature: 21°C, Humidity: 0.06
Temperature: 24°C, Humidity: 0.83
Temperature: 38°C, Humidity: 0.31
Temperature: 22°C, Humidity: 0.84
Temperature: 0°C, Humidity: 0.11
Temperature: 35°C, Humidity: 0.95
Temperature: 10°C, Humidity: 0.7
Temperature: 0°C, Humidity: 0.53
Temperature: 12°C, Humidity: 0.94
Temperature: 12°C, Humidity: 0.9
Temperature: 28°C, Humidity: 0.18
Temperature: 34°C, Humidity: 0.79
Temperature: 6°C, Humidity: 0.28
Temperature: 40°C, Humidity: 0.96
Temperature: 5°C, Humidity: 0.5
Temperature: 22°C, Humidity: 0.68
Temperature: 17°C, Humidity: 0.74
Temperature: 33°C, Humidity: 0.72
Temperature: 29°C, Humidity: 0.97
Temperature: 4°C, Humidity: 0.96
Temperature: 3°C, Humidity: 0.52
Temperature: 7°C, Humidity: 0.35
Temperature: 11°C, Humidity: 0.02
Temperature: 34°C, Humidity: 0.25
Temperature: 21°C, Humidity: 0.77
Temperature: 40°C, Humidity: 0.07
Temperature: 31°C, Humidity: 0.14
Temperature: 36°C, Humidity: 0.15
Temperature: 6°C, Humidity: 0.51
Temperature: 22°C, Humidity: 0.26
Temperature: 3°C, Humidity: 0.77

Result: Collecting sensor data from any real time application and apply preprocessing techniques is executed successfully.

EXP NO: 5	Collect sensor data and do Prediction using linear regression
Date:	

Aim: The aim is to Collect sensor data and do Prediction using linear regression.

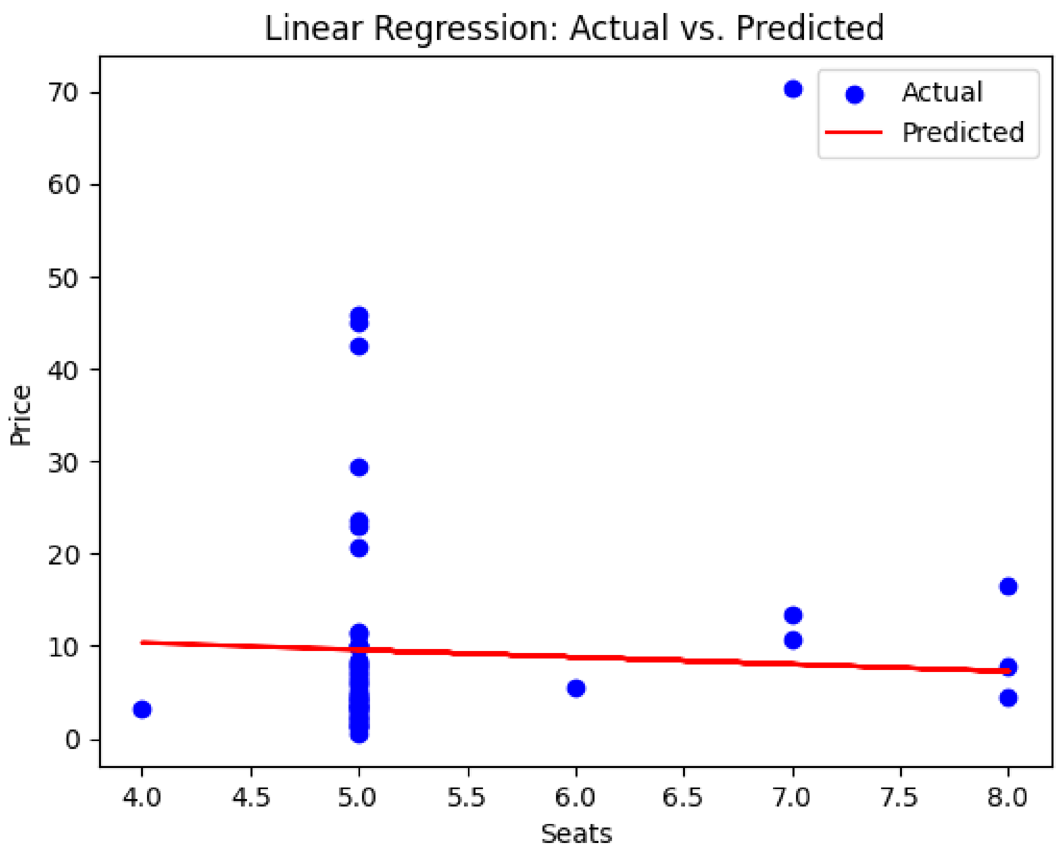
Procedure:

- We load the weather dataset using **pd.read_csv()** from **pandas**.
- We extract the humidity as the feature (**X**) and temperature as the target variable (**y**).
- We split the dataset into training and testing sets using **train_test_split** from **scikit-learn**.
- We produce relationship between one or more variables using Linear Regression.
- We train a model using a linear regression.
- We use the trained model to make predictions on the test data.
- Finally, we plot the actual vs. predicted values to visualize the performance of the Linear regression model.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load weather dataset
weather_data = pd.read_csv('used_cars_data1.csv')

X = weather_data[['Seats']]
y = weather_data['Price']

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train linear regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Make predictions
y_pred = lin_reg.predict(X_test)

# Plot the actual vs. predicted values
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted')
plt.xlabel('Seats')
plt.ylabel('Price')
plt.title('Linear Regression: Actual vs. Predicted')
plt.legend()
plt.show()
```

OUTPUT:



Result: Collecting sensor data and predicting using linear regression is executed successfully.

EXP NO: 6	Collect sensor data and Implement Support Vector Machine
Date:	

Aim: The aim is to collect sensor data from the IoT devices and Implement SVM for classification or prediction.

Procedure:

- We load the weather dataset using **pd.read_csv()** from **pandas**.
- We extract the humidity as the feature (**X**) and temperature as the target variable (**y**).
- We split the dataset into training and testing sets using **train_test_split** from **scikit-learn**.
- We standardize the features using **StandardScaler** to ensure that each feature has a mean of 0 and a standard deviation of 1.
- We train a Support Vector Machine (SVM) model with a linear kernel (**kernel='linear'**).
- We use the trained model to make predictions on the test data.
- Finally, we plot the actual vs. predicted values to visualize the performance of the SVM model.

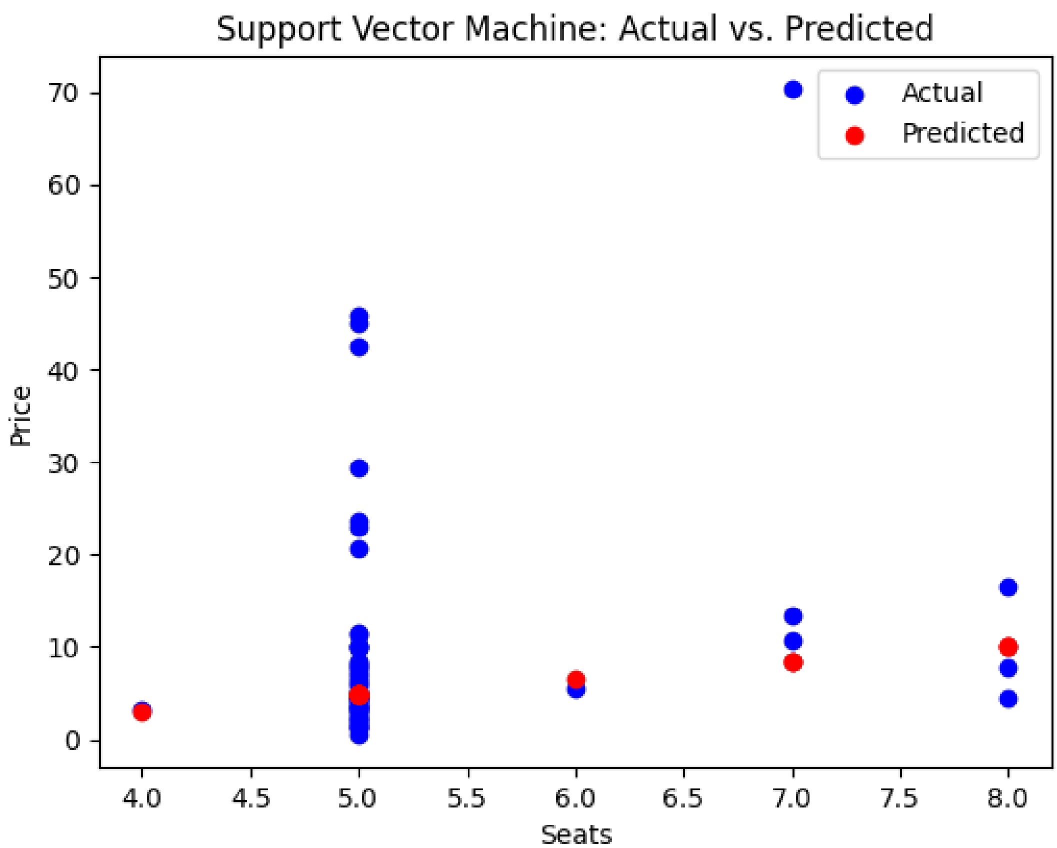
Note: Make sure to replace **'weather_data.csv'** with the path to your weather dataset CSV file.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load weather dataset
weather_data = pd.read_csv('used_cars_data1.csv')

X = weather_data[['Seats']]
y = weather_data['Price']

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Support Vector Machine (SVM) model
svm_model = SVR(kernel='linear')  # Linear kernel
svm_model.fit(X_train_scaled, y_train)
```

```
# Make predictions
y_pred = svm_model.predict(X_test_scaled)

# Plot the actual vs. predicted values
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.scatter(X_test, y_pred, color='red', label='Predicted')
plt.xlabel('Seats')
plt.ylabel('Price')
plt.title('Support Vector Machine: Actual vs. Predicted')
plt.legend()
plt.show()
```

OUTPUT:



Result: Collecting sensor data and Implementing Support Vector Machine is executed successfully.

EXP NO: 7	Collect sensor data and Implement Decision tree classification technique
Date:	

AIM: The aim is to collect sensor data and Implement Decision tree Classification.

Procedure:

- We load the weather dataset using **pd.read_csv()** from **pandas**.
- We define the features (**X**) as 'Temperature' and 'Humidity', and the target variable (**y**) as 'Weather'.
- We split the dataset into training and testing sets using **train_test_split** from **scikit-learn**.
- We train a Decision Tree classifier using **DecisionTreeClassifier**.
- We make predictions on the test data using the trained model.
- We evaluate the model's performance using accuracy, classification report, and confusion matrix.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load weather dataset
weather_data = pd.read_csv('used_cars_data1.csv')

# Define features (X) and target variable (y)
X = weather_data[['Price', 'Seats']]
y = weather_data['Transmission']

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

# Make predictions
y_pred = dt_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Display confusion matrix
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

OUTPUT:

```
Accuracy: 0.7627118644067796
Classification Report:
              precision    recall  f1-score   support

   Automatic       0.61      0.61      0.61        18
      Manual       0.83      0.83      0.83        41

    accuracy                           0.76        59
   macro avg       0.72      0.72      0.72        59
weighted avg       0.76      0.76      0.76        59

Confusion Matrix:
[[11  7]
 [ 7 34]]
```

Result: Collecting sensor data and Implementing Decision tree classification technique is executed successfully.

EXP NO: 8	Collect sensor data and Implement clustering algorithm
Date:	

AIM: The aim is to collect sensor data and Implement clustering algorithm.

Procedure:

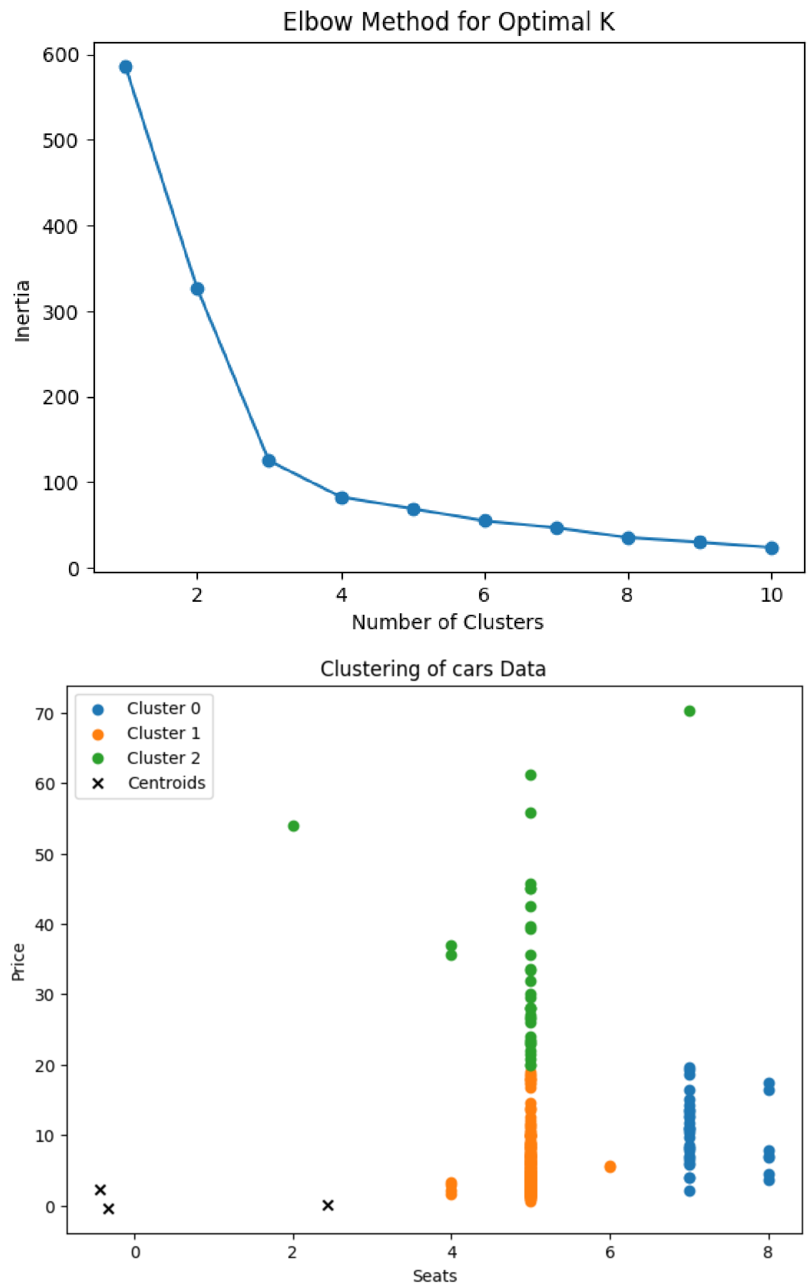
- We load the weather dataset using **pd.read_csv()**.
- We select features such as temperature and humidity.
- We standardize the features using **StandardScaler** to ensure that each feature has a mean of 0 and a standard deviation of 1.
- We use the Elbow method to determine the optimal number of clusters.
- Based on the Elbow method, we choose the optimal number of clusters.
- We apply KMeans clustering with the chosen number of clusters.
- We add cluster labels to the dataset.
- Finally, we plot the clusters and centroids using matplotlib.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load weather dataset
weather_data = pd.read_csv('used_cars_data1.csv')

# Select features (e.g., Temperature and Humidity)
X = weather_data[[ 'Seats','Price']]

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Determine the optimal number of clusters using the Elbow method
inertia = []
for n_clusters in range(1, 11):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow method to determine the optimal number of clusters
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.show()
```

```
# Based on the Elbow method, let's choose the optimal number of clusters (e.g., 3 or 4)

# Apply KMeans clustering
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(X_scaled)
labels = kmeans.labels_
centers = kmeans.cluster_centers_

# Add cluster labels to the dataset
weather_data['Cluster'] = labels

# Plot the clusters
plt.figure(figsize=(8, 6))
for cluster in range(n_clusters):
    cluster_data = weather_data[weather_data['Cluster'] == cluster]
    plt.scatter(cluster_data['Seats'], cluster_data['Price'], label=f'Cluster {cluster}')
plt.scatter(centers[:, 0], centers[:, 1], color='black', marker='x', label='Centroids')
plt.xlabel('Seats')
plt.ylabel('Price')
plt.title('Clustering of cars Data')
plt.legend()
plt.show()
```

OUTPUT:



Result: Collecting sensor data and Implementing clustering algorithm is executed successfully.

EXP NO: 9	Visualize data using visualization techniques
Date:	

AIM: The aim is to visualize data using visualization techniques.

Procedure:

- We load the weather dataset using **pd.read_csv()** from **pandas**.
- We display the first few rows of the dataset and summary statistics of numerical variables using **head()** and **describe()** functions, respectively.
- We visualize the distribution of temperature and humidity using histograms.
- We create a scatter plot of temperature vs. humidity to explore their relationship.
- We plot box plots to visualize the distribution of temperature for different weather conditions.
- We use a pairplot to visualize pairwise relationships between different variables in the dataset.

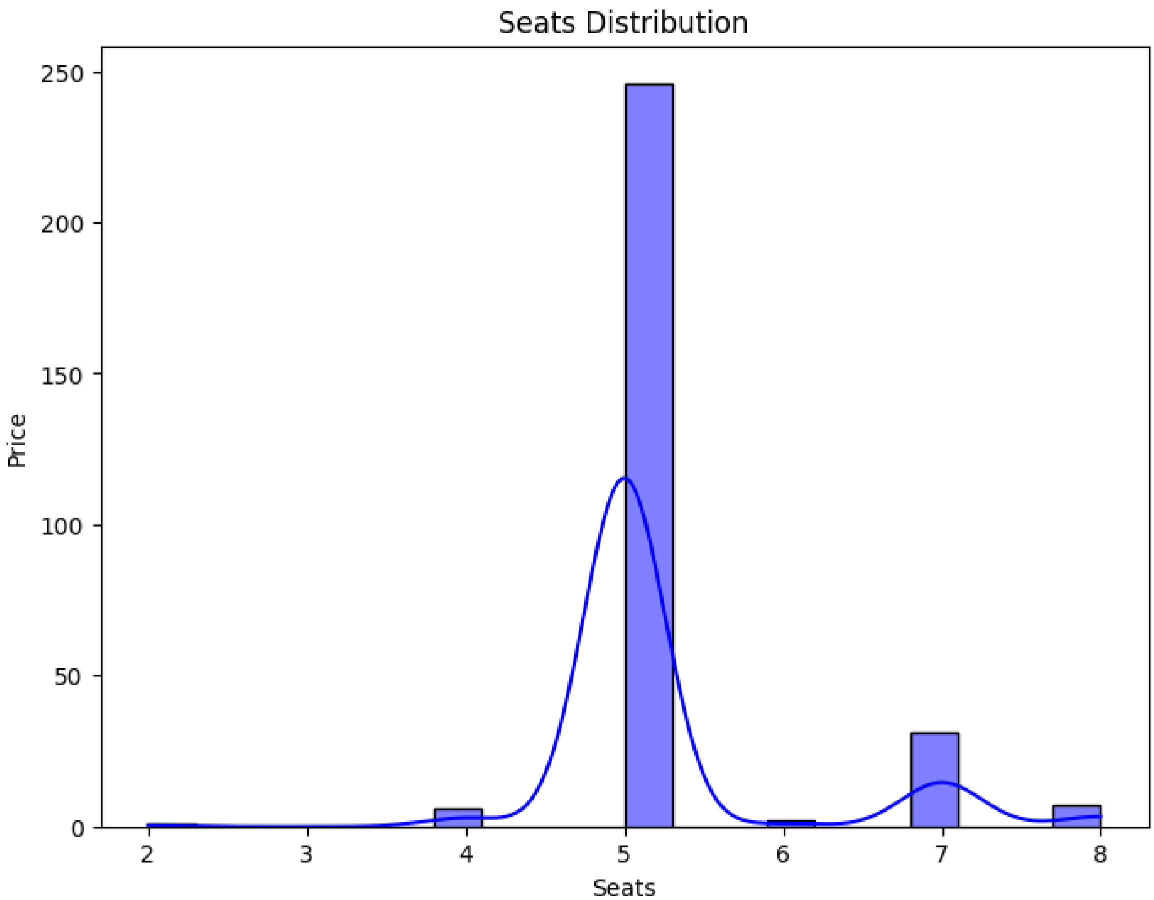
Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
weather_data = pd.read_csv('used_cars_data1.csv')
print("First few rows of the dataset:")
print(weather_data.head())
print("\nSummary statistics of numerical variables:")
print(weather_data.describe())

plt.figure(figsize=(8, 6))
sns.histplot(weather_data['Seats'], bins=20, kde=True, color='blue')
plt.xlabel('Seats')
plt.ylabel('Price')
plt.title('Seats Distribution')
plt.show()

plt.figure(figsize=(8, 6))
sns.histplot(weather_data['Price'], bins=20, kde=True, color='green')
plt.xlabel('Price')
plt.ylabel('rev')
plt.title('Price Distribution')
plt.show()
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Seats', y='rev', data=weather_data, color='red')
plt.xlabel('Seats')
plt.ylabel('rev')
plt.title('Seats vs. rev')
plt.show()
```

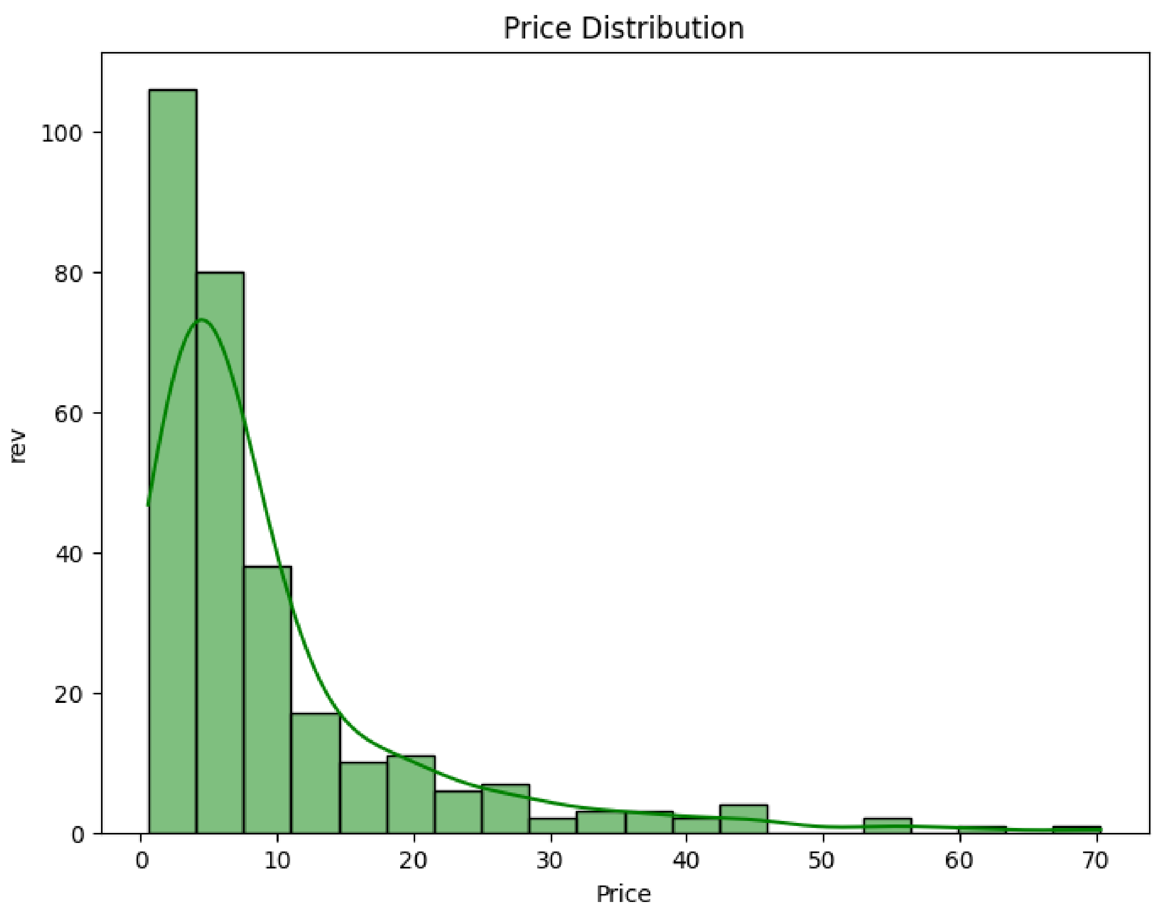
```
plt.figure(figsize=(10, 6))
sns.boxplot(x='rev', y='Seats', data=weather_data)
plt.xlabel('rev')
plt.ylabel('Seats')
plt.title('Seats by rev')
plt.show()
sns.pairplot(weather_data, diag_kind='kde')
plt.suptitle('Pairwise Relationships')
plt.show()
```

OUTPUT:

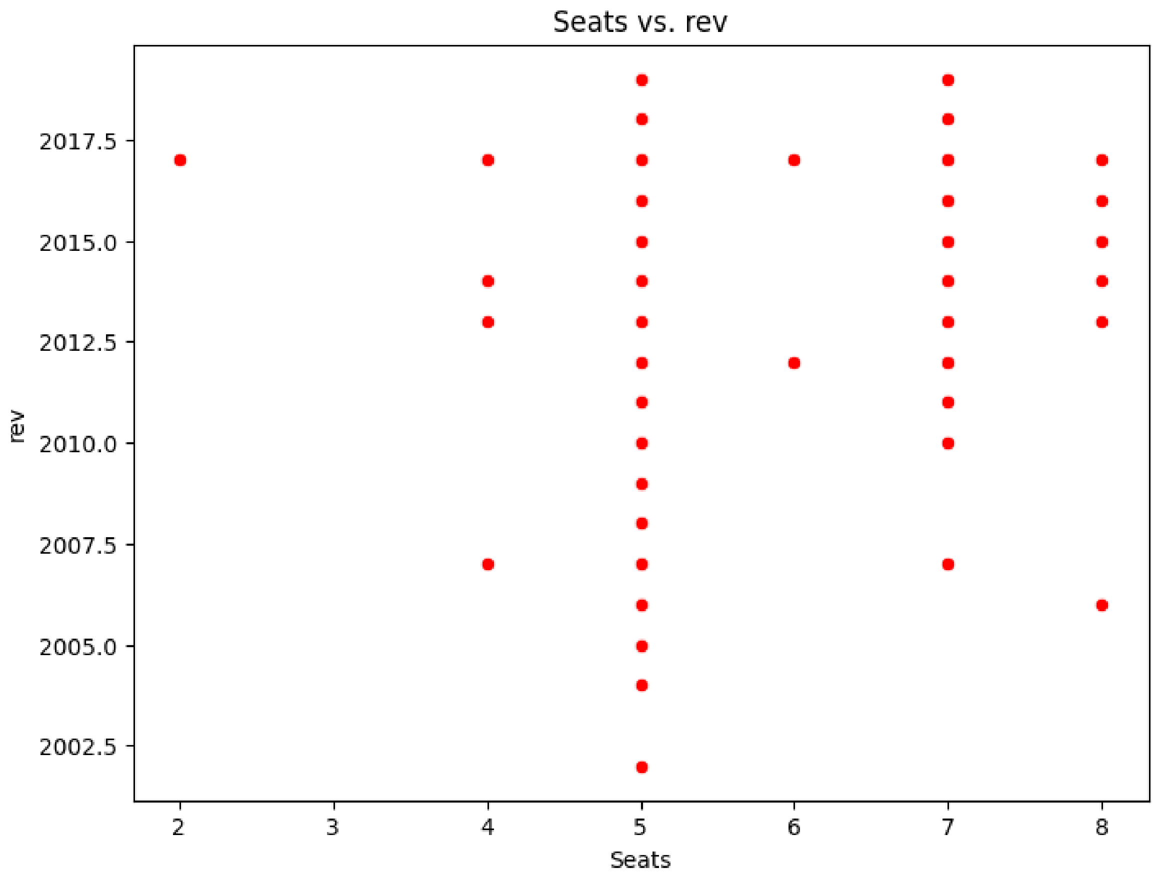
```
First few rows of the dataset:
   S.No.                              Name    Location   rev  \
0      0            Maruti Wagon R LXI CNG      Mumbai  2010
1      1  Hyundai Creta 1.6 CRDi SX Option        Pune  2015
2      2                      Honda Jazz V     Chennai  2011
3      3                 Maruti Ertiga VDI     Chennai  2012
4      4   Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013

   Kilometers_Driven Fuel_Type Transmission Owner_Type     Mileage   Engine  \
0              72000       CNG       Manual      First  26.6 km/kg   998 CC
1              41000    Diesel       Manual      First  19.67 kmpl  1582 CC
2              46000    Petrol       Manual      First   18.2 kmpl  1199 CC
3              87000    Diesel       Manual      First  20.77 kmpl  1248 CC
4              40670    Diesel    Automatic     Second   15.2 kmpl  1968 CC

       Power  Seats  New_Price  Price
0  58.16 bhp      5        NaN   1.75
1  126.2 bhp      5        NaN  12.50
2   88.7 bhp      5  8.61 Lakh   4.50
3  88.76 bhp      7        NaN   6.00
4  140.8 bhp      5        NaN  17.74

Summary statistics of numerical variables:
            S.No.          rev  Kilometers_Driven       Seats       Price
count  293.000000   293.000000         293.000000  293.000000  293.000000
mean   146.000000  2013.351536       56543.812287    5.259386    9.504949
std     84.726029     3.148598       37089.072104    0.794430   10.679840
min      0.000000  2002.000000        1000.000000    2.000000    0.550000
25%     73.000000  2012.000000       32700.000000    5.000000    3.350000
50%    146.000000  2014.000000       52000.000000    5.000000    5.580000
75%    219.000000  2016.000000       71088.000000    5.000000   10.500000
max    292.000000  2019.000000      262000.000000    8.000000   70.360000
```

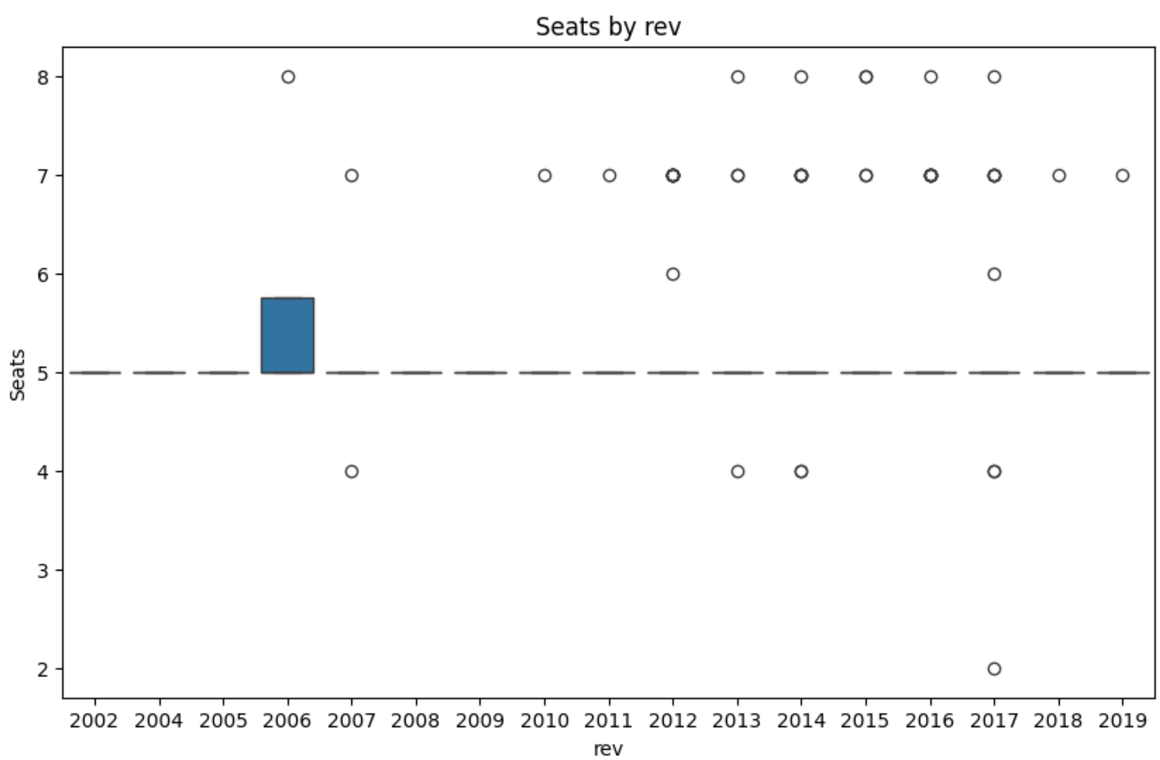


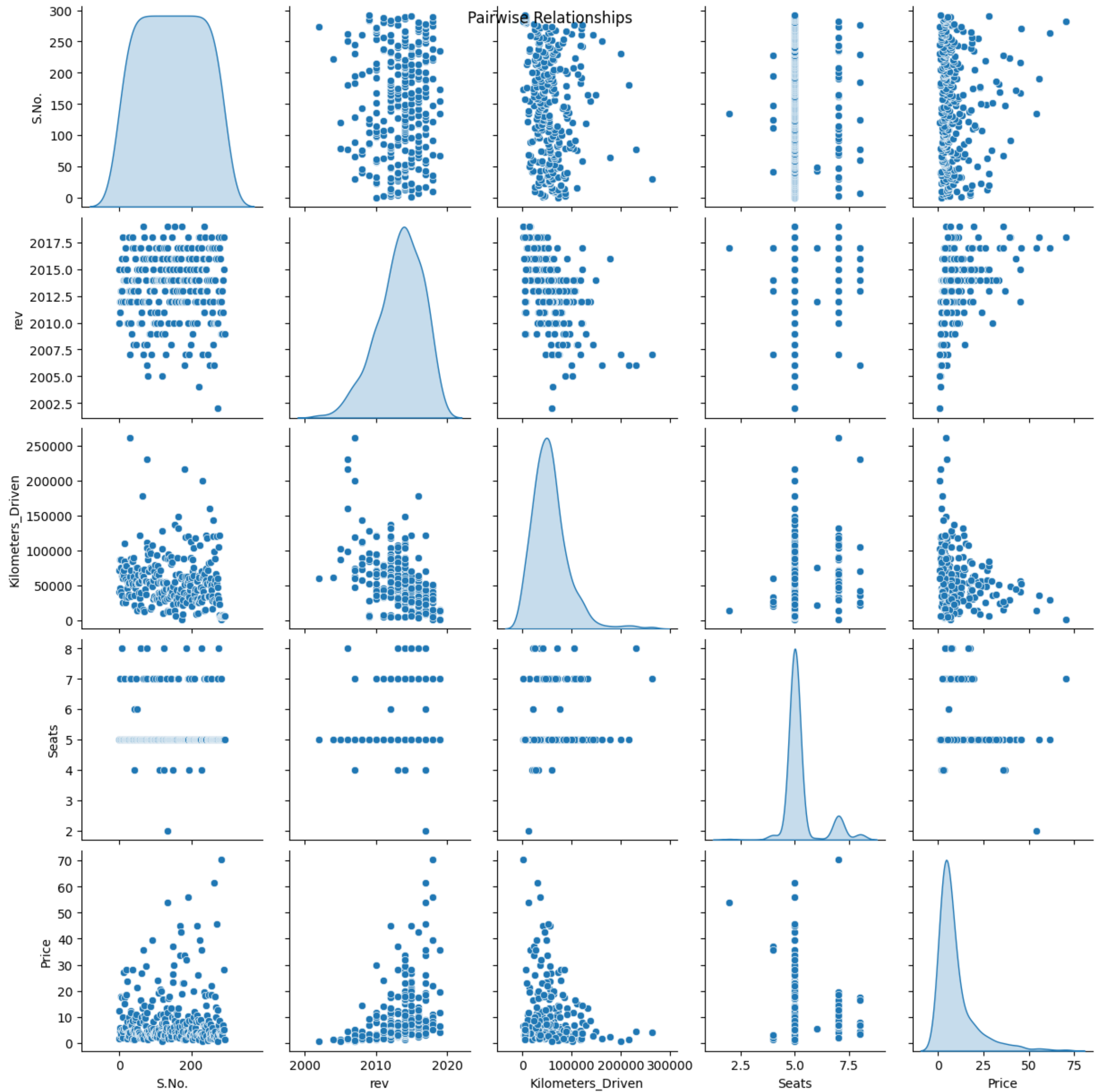
Seats Distribution
Price
250
200
150
100
50
0
2
3
4
5
6
7
8
Seats



Price Distribution
rev
100
80
60
40
20
0
0
10
20
30
40
50
60
70
Price


Seats vs. rev
2017.5
2015.0
2012.5
2010.0
2007.5
2005.0
2002.5
rev
2
3
4
5
6
7
8
Seats


Seats by rev
8
7
6
5
4
3
2
Seats
2002
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
rev



Result: Visualizing data using visualization techniques is executed successfully.

EXP NO: 10	Model Time series data
Date:	

AIM: The aim is to analyze the Time series data by using ARIMA Model.

Procedure:

Modeling time series data involves analyzing and forecasting data points based on their temporal order. One popular method for time series forecasting is using Autoregressive Integrated Moving Average (ARIMA) models.

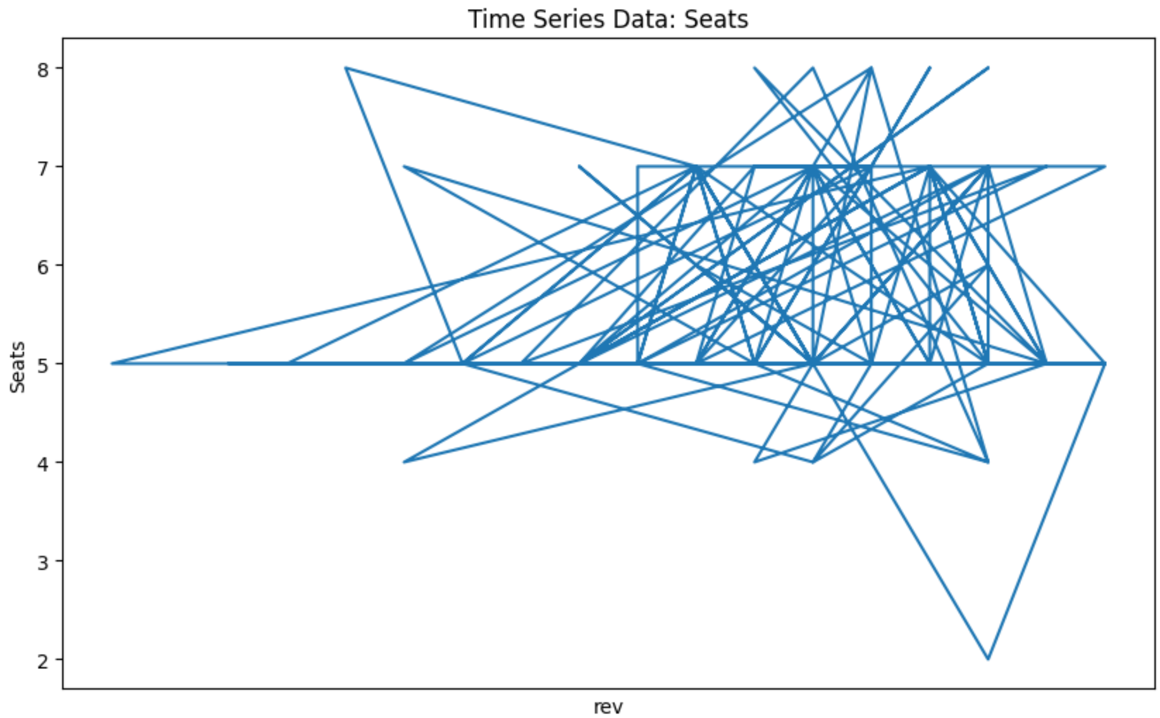
- We load the time series data from a CSV file using **pd.read_csv()** from **pandas**.
- We convert the 'Date' column to datetime format and set it as the index of the DataFrame.
- We plot the time series data to visualize its pattern and trends.
- We plot autocorrelation and partial autocorrelation plots to determine the appropriate parameters for the ARIMA model.
- We fit an ARIMA model to the time series data using the specified order **(p, d, q)**.
- We print the summary of the ARIMA model to examine its coefficients and statistical information.
- We plot the residuals of the model to check for any patterns or trends.
- We forecast future values using the trained ARIMA model and plot the original data along with the forecasted values.

Code:

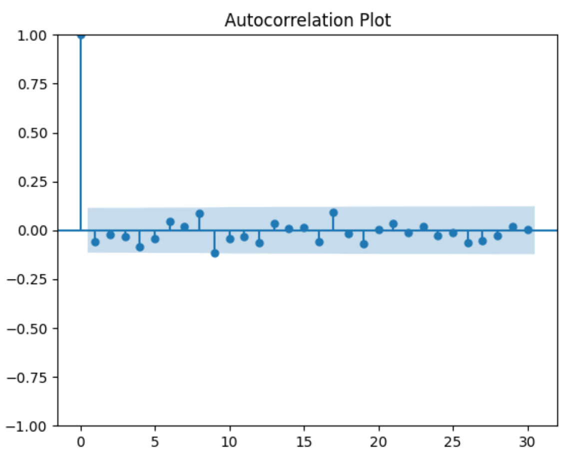
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Load time series data
data = pd.read_csv('used_cars_data1.csv')

# Convert the 'date' column to datetime format and set it as the index
data['rev'] = pd.to_datetime(data['rev'])

# Drop rows with missing dates
data.dropna(subset=['rev'], inplace=True)

# Set the index to the 'date' column
data.set_index('rev', inplace=True)

# Select only the first 400 columns for analysis
```

```
data_selected = data.iloc[:, :400]

# Plot original 'BMI' time series data against selected dates
plt.figure(figsize=(10, 6))
plt.plot(data_selected.index, data_selected['Seats'])  # Plotting 'BMI' against selected dates
plt.title('Time Series Data: Seats')
plt.xlabel('rev')
plt.ylabel('Seats')
plt.show()

# Plot autocorrelation and partial autocorrelation plots for 'BMI' column
plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plot_acf(data_selected['Seats'], lags=30, ax=plt.gca())
plt.title('Autocorrelation Plot')
plt.subplot(1, 2, 2)
plot_pacf(data_selected['Seats'], lags=30, ax=plt.gca())
plt.title('Partial Autocorrelation Plot')
plt.show()

# Fit ARIMA model
order = (2, 1, 1)  # (p, d, q)
model = ARIMA(data_selected['Seats'], order=order)
result = model.fit()
print(result.summary())

plt.figure(figsize=(10, 6))
plt.plot(result.resid)
plt.title('Model S.No.')
plt.xlabel('rev')
plt.ylabel('S.No.')
plt.show()

# Forecast future values
forecast_steps = 12  # Number of steps to forecast
forecast = result.forecast(steps=forecast_steps)
plt.figure(figsize=(10, 6))
plt.plot(data_selected.index, data_selected['Seats'], label='Original Data')
plt.plot(pd.date_range(start=data_selected.index[-1], periods=forecast_steps+1,
freq='M')[1:], forecast, label='Forecasted Values', linestyle='--')
plt.title('Original Data vs Forecasted Values')
plt.xlabel('rev')
plt.ylabel('Seats')
plt.legend()
plt.show()
```

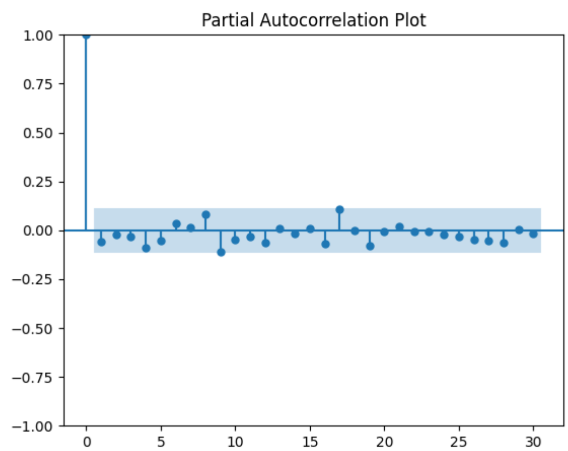
OUTPUT:



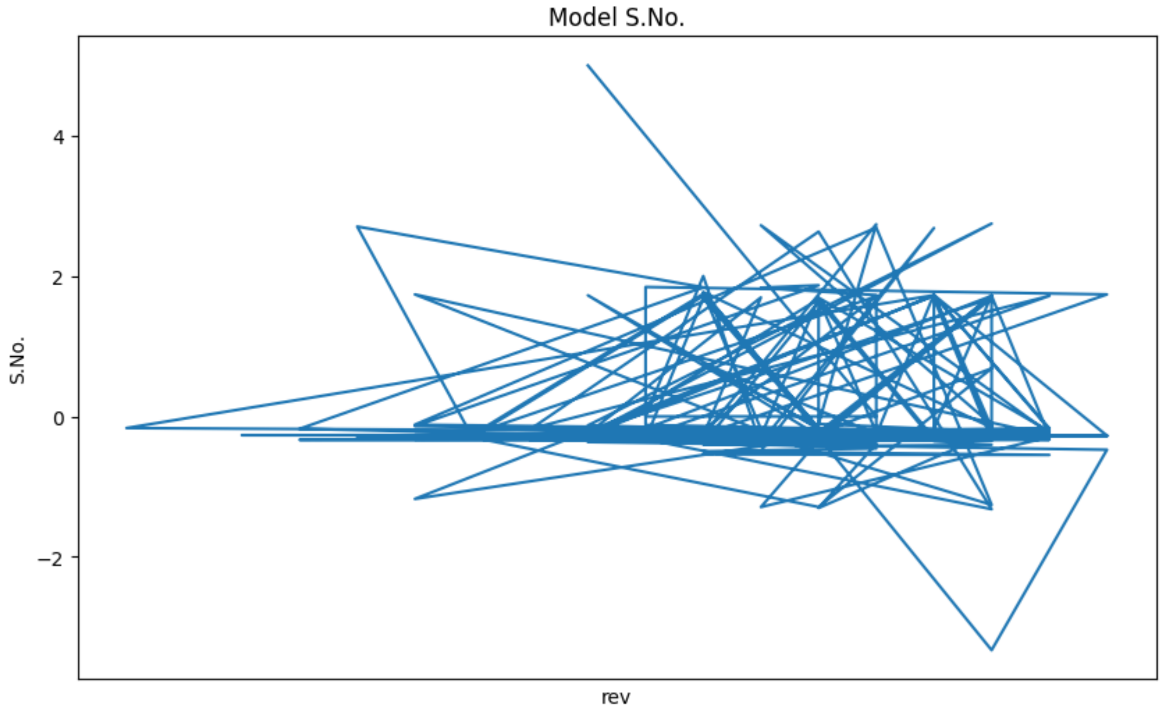
Time Series Data: Seats
Seats
rev
8
7
6
5
4
3
2

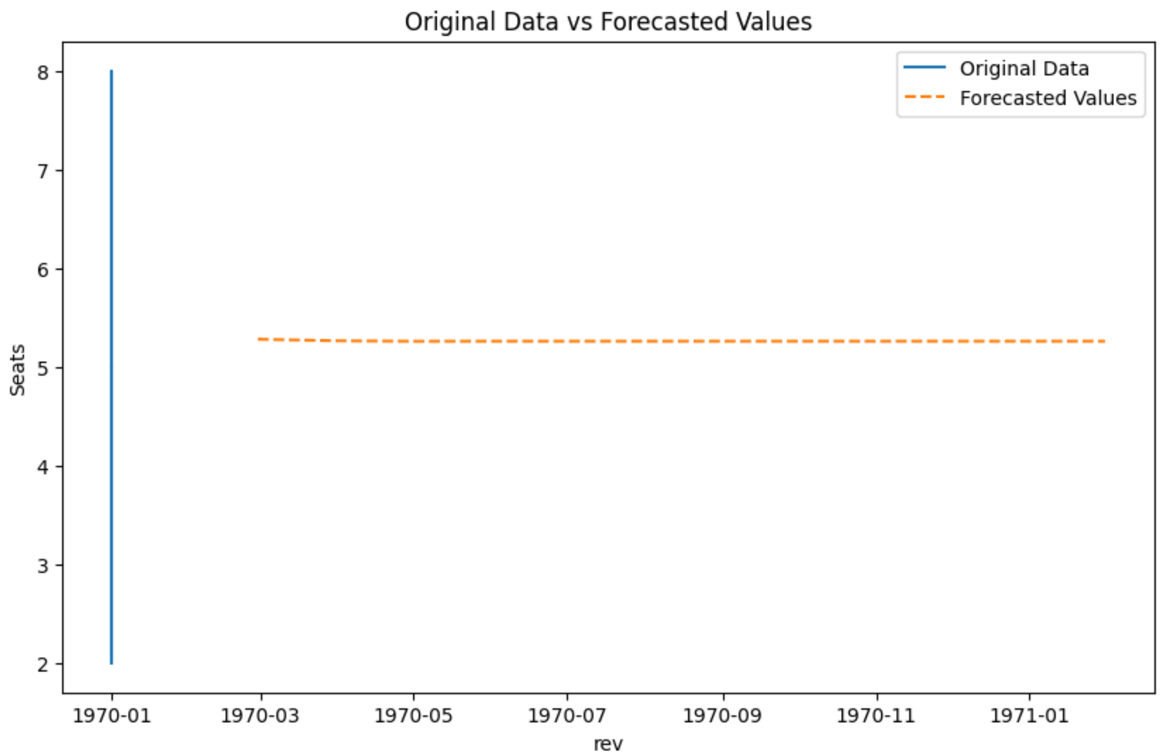


Autocorrelation Plot
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30



Partial Autocorrelation Plot
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30





Result: Modeling time series data involves analyzing and forecasting data points based on their temporal order is executed successfully.

EXP NO: 11	Implement an application that stores big data in HBase/ MongoDB/ Pig
Date:	

Aim: Aim to implement an application that stores big data in Hbase/ MongoDB/ Pig.

Procedure:

1. **Installation**:

   - First, ensure you have access to a MongoDB database. You can download a free MongoDB database from here or use a MongoDB cloud service like MongoDB Atlas.
   - Next, install the **PyMongo** driver using pip. If you haven't already, open your command line and run the following command:

   - python -m pip install pymongo

2. **Test PyMongo**:

   - To verify that the installation was successful, create a Python file (let's call it demo_mongodb_test.py) with the following content:

     Python

     ```
     # demo_mongodb_test.py
     import pymongo
     # Test if pymongo is installed
     print("PyMongo is installed and ready to be used.")
     ```

   - Execute the above code. If no errors occur, you're all set to use PyMongo!

3. **Basic CRUD Operations**:

   - With PyMongo, you can perform the following operations:

     1. **Create**: Insert data into MongoDB.
     2. **Read**: Retrieve data from MongoDB.
     3. **Update**: Modify existing data.
     4. **Delete**: Remove data from MongoDB.

Example Usage:

Here’s a simple example of inserting data into a MongoDB collection:

```
import pymongo

# Connect to MongoDB

client =pymongo.MongoClient("mongodb://localhost:27017/")

db = client["mydatabase"]

collection = db["mycollection"]

# Insert a document

data = {"name": "John", "age": 30}

collection.insert_one(data)
```

OUTPUT:

Succesful Insertion: ObjectId(‘6328d347dfy7e82rh34m089zlp253’)

Data inserted Succsfully.

Result: Implementing an application that stores big data in HBase/ MongoDB/Pig is executed succesfully.

EXP NO: 12	**Implement an application for predicting air pollution level using gas sensors.**
Date:	

Aim: The aim is to Implement an application for predicting air pollution level using gas sensors.

Procedure:

Step 1: Prepare Your Environment

First, ensure you have the necessary libraries installed. If not, install them using pip:

```
pip install numpy pandas scikit-learn matplotlib
```

Step 2: Sample Dataset

Imagine we have a CSV file named **air_quality.csv** with sensor readings for CO, NO2, and O3, alongside the target variable PM2.5 (particulate matter size 2.5 which is a common measure for air pollution levels).

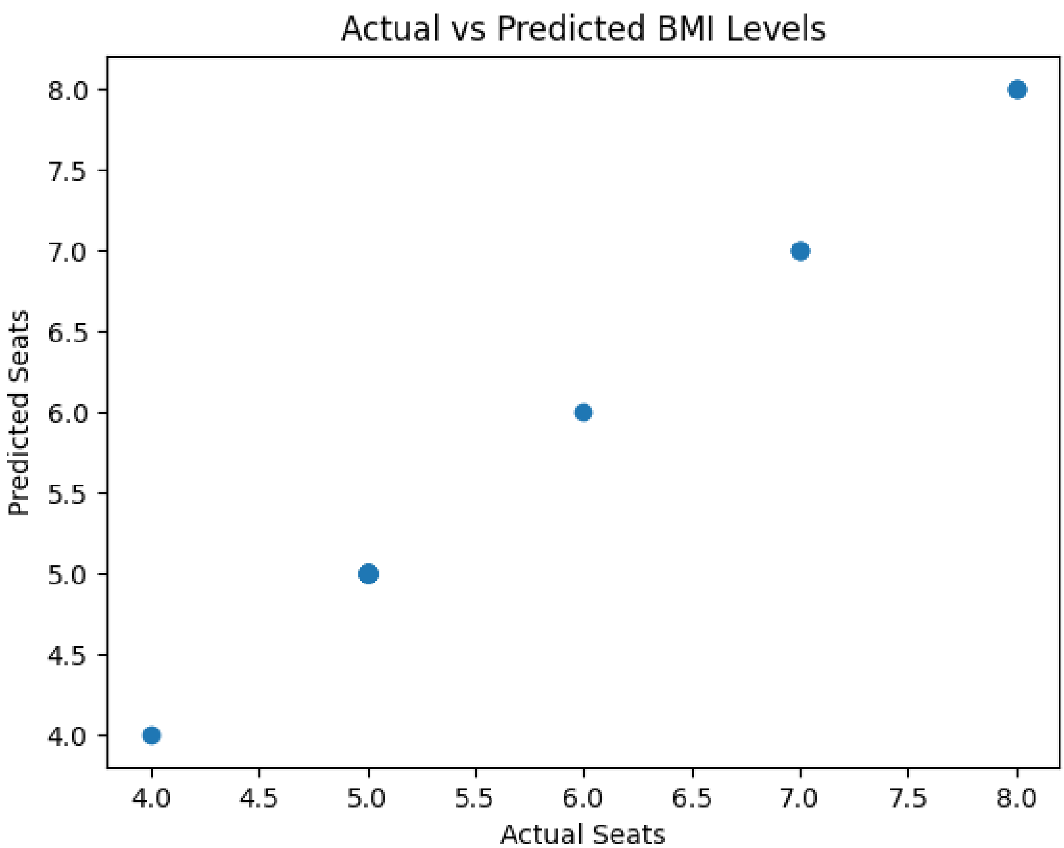
```
CO,NO2,O3,PM2.5
0.4,0.02,0.03,12
0.25,0.01,0.02,9
0.5,0.03,0.04,15
...
```

Python Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('used_cars_data1.csv')

# Select features and target
X = df[['Price', 'Seats', 'rev']]  # Features: Sensor readings
y = df['Seats']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

```
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")

plt.scatter(y_test, y_pred)
plt.xlabel("Actual Seats")
plt.ylabel("Predicted Seats")
plt.title("Actual vs Predicted BMI Levels")
plt.show()
```

OUTPUT:



Result: Implementing an application for predicting air pollution level using gas sensors is executed successfully.